UDACITY

PROJECT

Advanced Lane Finding

A part of the Self Driving Car Engineer Nanodegree Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Writeup / README

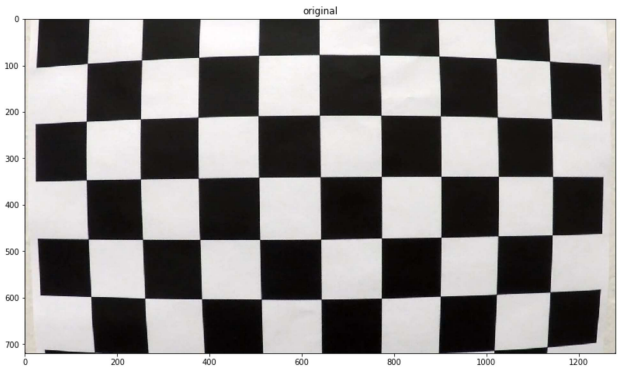
The writeup / README should include a statement and supporting figures / images that explain how each rubric item was addressed, and specifically where in the code each step was handled.

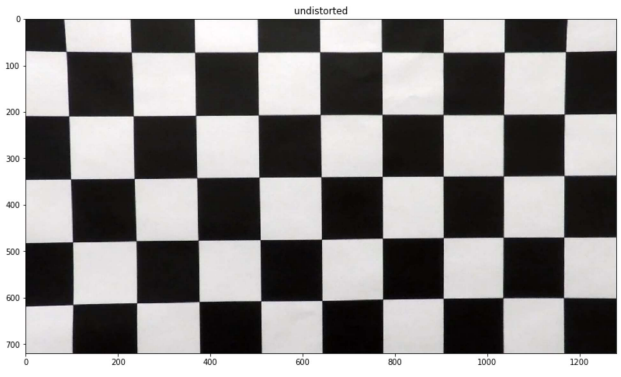
Good job addressing the rubric items in the writeup.md file with references to supporting images.

Camera Calibration

OpenCV functions or other methods were used to calculate the correct camera matrix and distortion coefficients using the calibration chessboard images provided in the repository (note these are 9x6 chessboard images, unlike the 8x6 images used in the lesson). The distortion matrix should be used to un-distort one of the calibration images provided as a demonstration that the calibration is correct. Example of undistorted calibration image is Included in the writeup (or saved to a folder).

Well done finding the Chessboard corners with **cv2.findChessboardCorners** and calculating the camera matrix and distortion coefficients with **cv2.calibrateCamera**. Good job undistorting the chessboard image with **cv2.undistort** and outputting the undistorted image as shown below.





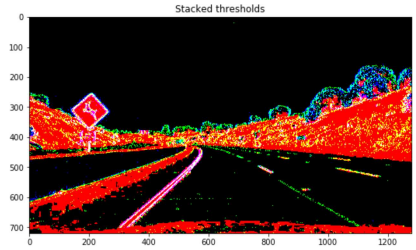
Pipeline (test images)

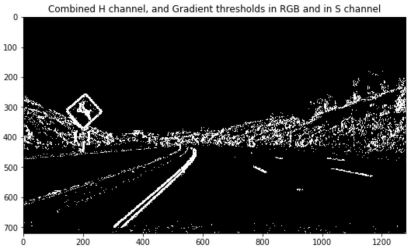
Distortion correction that was calculated via camera calibration has been correctly applied to each image. An example of a distortion corrected image should be included in the writeup (or saved to a folder) and submitted with the project.

Good job using **cv2.undistort** to undistort the test images. Below is one of the outputted test images from the 'P4.ipynb' file.

A method or combination of methods (i.e., color transforms, gradients) has been used to create a binary image containing likely lane pixels. There is no "ground truth" here, just visual verification that the pixels identified as part of the lane lines are, in fact, part of the lines. Example binary images should be included in the writeup (or saved to a folder) and submitted with the project.

Well done using the saturation channel, hue channel, and a sobel x operator to output the final thresholded binary image. Below is one of the outputted images from the 'P4.ipynb' file.

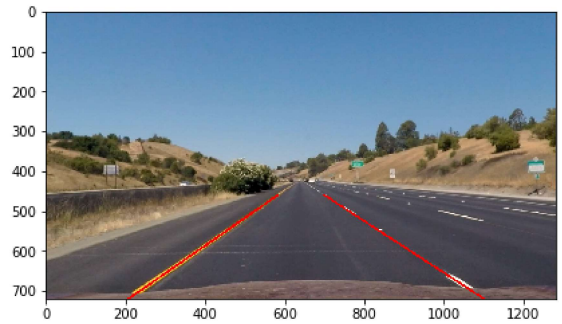


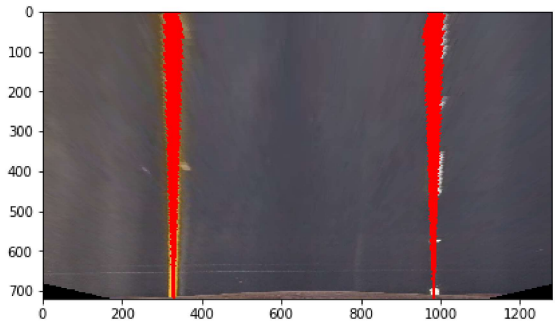


OpenCV function or other method has been used to correctly rectify each image to a "birds-eye view". Transformed images should be included in the writeup (or saved to a folder) and submitted with the project.

Good job marking the source and desired coordinate points. Well done finding the perspective transform with **cv2.getPerspectiveTransform** and using **cv2.warpPerspective** to warp the image into a "birds-eye view".

The warped images looks good. Well done marking the source points on a straight lane image. The warped lanes should be near vertical because the lanes should be straight if not nearly straight. Below is one of the outputted images from the 'P4.ipynb' file.





Methods have been used to identify lane line pixels in the rectified binary image. The left and right line have been identified and fit with a curved functional form (e.g., spine or polynomial). Example images with line pixels identified and a fit overplotted should be included in the writeup (or saved to a folder) and submitted with the project.

Good job initializing the histogram method with:

```
histogram = np.sum(binary_warped[binary_warped.shape[0]//3:,:], axis=0)
```

Well done finding the histogram peaks with np.argmax().

Well done implementing a sliding window with a margin width of 80 pixels wide and the maximum number of pixels to recenter the sliding window set to 50.

Good job using np.polyfit() to find the polynomial coefficients and finding the x coordinates of the polynomial fit with the left_fitx and right_fix variables.

Good job using the previously fitted lane curve as a starting point to find the current curves. Well done using a x coordinate margin of 80 to find the new lane pixels and np.polyfit() to find the new coefficients of the curves.

Nice job setting up a sanity check with the code below:

```
if lastFrame.flg == True:
    # Widt of the lane: No more then 15%
    if (abs(width_lane-lastFrame.width_lane)/lastFrame.width_lane) > 0.15:
        error = 1
    # Car position: No more than 1m
    if abs(car_position-lastFrame.car_position) > 1:
        error = 1
    # Difference betwee left and right raidus less than 50%
    if abs(right_curverad/left_curverad) < 0.5:
        error = 1
    if abs(left_curverad/right_curverad) < 0.5:
        error = 1
```

Here the idea is to take the measurements of where the lane lines are and estimate how much the road is curving and where the vehicle is located with respect to the center of the lane. The radius of curvature may be given in meters assuming the curve of the road follows a circle. For the position of the vehicle, you may assume the camera is mounted at the center of the car and the deviation of the midpoint of the lane from the center of the image is the offset you're looking for. As with the polynomial fitting, convert from pixels to meters.

Well done finding the radius of curvature values with:

```
left_curverad = ((1 + (2*left_fit_cr[0]*y_eval*ym_per_pix + left_fit_cr[1])**2)**1.5) / np.absolute(2*left_fit_cr[0])
right_curverad = ((1 + (2*right_fit_cr[0]*y_eval*ym_per_pix + right_fit_cr[1])**2)**1.5) / np.absolute(2*right_fit_cr[0])
```

Good job converting the units from pixels to meters with:

```
ym_per_pix = 30/720 # meters per pixel in y dimension
xm_per_pix = 3.7/653 # meters per pixel in x dimension
```

Good job calculating the deviation from center value with:

```
center_lane =  ((right_fitx[out_img.shape[0]-1]+left_fitx[out_img.shape[0]-1])/2)
car_position = (center_lane - out_img.shape[1]/2)*xm_per_pix
```

Well done using cv2.putText to output the radius of curvature values and the deviation from center value.

The fit from the rectified image has been warped back onto the original image and plotted to identify the lane boundaries. This should demonstrate that the lane boundaries were correctly identified. An example image with lanes, curvature, and position from center should be included in the writeup (or saved to a folder) and submitted with the project.

Good job using **cv2.warpPerspective** to unwarp the lane area image back into the original image space and combine it with the original undistorted image with **cv2.addWeighted**.

Pipeline (video)

The image processing pipeline that was established to find the lane lines in images successfully processes the video. The output here should be a new video where the lanes are identified in every frame, and outputs are generated regarding the radius of curvature of the lane and vehicle position within the lane. The pipeline should correctly map out curved lines and not fail when shadows or pavement color changes are present. The output video should be linked to in the writeup and/or saved and submitted with the project.

Well done outputting the annotated video. It looks good.

Discussion

Discussion includes some consideration of problems/issues faced, what could be improved about their algorithm/pipeline, and what hypothetical cases would cause their pipeline to fail.

Nice job mentioning in the discussion, "The pipeline fails when another vertical lines appear in the road: for example when the asphalt has two different colors, black wheel marks...
I solved this with the sanity check. "

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

Student FAQ Reviewer Agreement